MTS: Bringing Multi-Tenancy to Virtual Networking

Kashyap Thimmaraju[1] Saad Hermak[1] Gábor Rétvári[2] Stefan Schmid[3]
[1] *Technische Universität Berlin* [2] *BME HSNLab* [3] *Faculty of Computer Science, University of Vienna*

Abstract

Multi-tenant cloud computing provides great benefits in terms of resource sharing, elastic pricing, and scalability, however, it also changes the security landscape and introduces the need for strong isolation between the tenants, *also inside the network*. This paper is motivated by the observation that while multi-tenancy is widely used in cloud computing, the virtual switch designs currently used for network virtualization lack sufficient support for tenant isolation. Hence, we present, implement, and evaluate a virtual switch architecture, *MTS*, which brings secure design best-practice to the context of multi-tenant virtual networking: compartmentalization of virtual switches, least-privilege execution, complete mediation of all network communication, and reducing the trusted computing base shared between tenants. We build *MTS* from commodity components, providing an incrementally deployable and inexpensive upgrade path to cloud operators. Our extensive experiments, extending to both micro-benchmarks and cloud applications, show that, depending on the way it is deployed, *MTS* may produce 1.5-2x the throughput compared to state-of-the-art, with similar or better latency and modest resource overhead (1 extra CPU). *MTS* is available as open source software.

1 Introduction

Security landscape of cloud virtual networking. Datacenters have become a critical infrastructure of our digital society, and with the fast growth of data centric applications and AI/ML workloads, dependability requirements on cloud computing will further increase [13]. At the heart of an efficiently operating datacenter lies the idea of resource sharing and *multi-tenancy*: independent instances (e.g., applications or tenants) can utilize a given infrastructure concurrently, including the compute, storage, networking, and management resources deployed at the data center, in a physically integrated but logically *isolated* manner [37, 22].

At the level of the data center communication network, isolation is provided by the network virtualization architecture. Key to network virtualization is the *virtual switch (vswitch)*, a network component located in the Host virtualization layer of the (edge) servers that connects tenants' compute and storage resources (e.g., Virtual Machines (VMs), storage volumes, etc.), provisioned at the server, to the rest of the data center and the public Internet [37, 32, 52].

Multi-tenancy is typically provided in this design by (i) deploying the vswitches with the server's Host operating system/hypervisor (e.g, Open vSwitch aka OvS [54]); (ii) using *flow-table-level isolation*: the vswitch's flow tables are divided into per-tenant logical datapaths that are populated with sufficient flow table entries to link tenants' data-center-bound resources into a common interconnected workspace [37, 32, 52]; and (iii) overlay networks using a tunneling protocol, e.g., VXLAN [70], to connect tenants' resources into a single workspace. Alternatives to this Host-based vswitch model [54], e.g., NIC-based vswitch solutions [34, 26] and FPGA-based designs [21], share the main trait that the logical datapaths have a common networking substrate (vswitch).

Despite the wide-scale deployment [16, 21, 34], the level of (logical and performance) isolation provided by vswitches is not yet well-understood. For example, Thimmaraju et al. [66] uncovered a serious isolation problem with a popular virtual switch (OvS). An adversary could not only break out of the VM and attack all applications on the Host, but could also manifest as a worm, and compromise an entire datacenter in a few minutes. Csikor et al. [15] identified a severe performance isolation vulnerability, also in OvS, which results in a low-resource cross-tenant denial-of-service attack. Such attacks may exacerbate concerns surrounding the security and adoption of public clouds (that is already a major worry across cloud users [60]).

Indeed, a closer look at the cloud virtual networking best-practice, whereby *per-tenant logical datapaths are deployed on a single Host-based vswitch using flow-table-level isolation* [37, 32, 52], reveals that the current state-of-the-art violates basically all relevant secure system design princi-

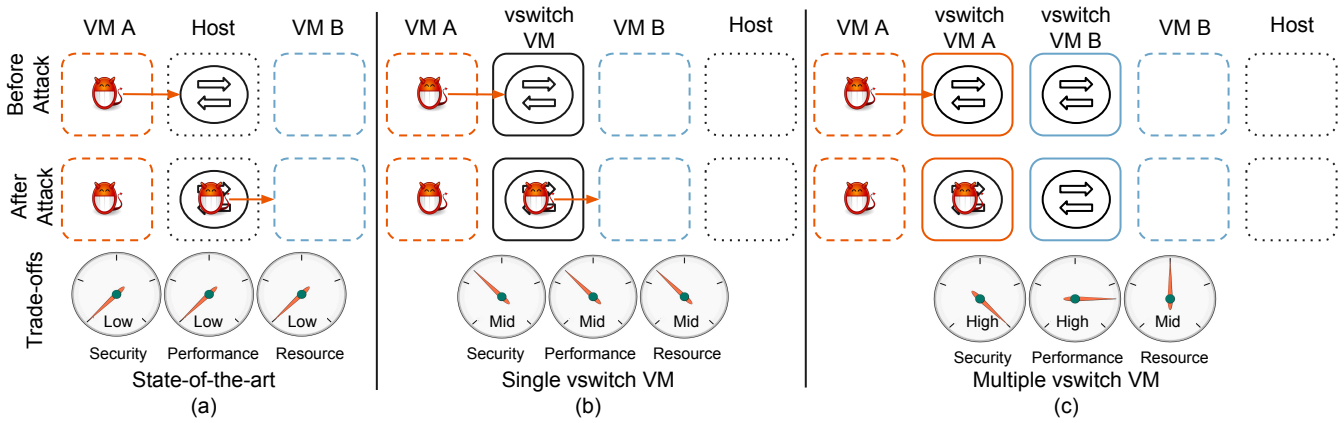

Figure 1: A high-level view of the tradeoffs between security, performance and resources for the state-of-the-art and *MTS*

ples [59, 9]. First, the principle of *least privilege* would require that any system component should be given only the minimum set of privileges necessary to complete its task, yet we see that vswitch code typically executes *on the Host* with administrator, or what is worse, with full kernel privilege [69, 27], even though this would not be absolutely necessary (see Sec. 3). Second, untrusted user code directly interacts with the vswitch and hence with the Host OS, e.g., it may send arbitrary packets from VMs, query statistics, or even install flow table entries through side channels [15], which violates the secure design principle of *complete mediation*. But most importantly, the shared vswitch design goes directly against the principle of the *least common mechanism*, which would minimize the amount of resources common to more than one tenant.

Secure vswitch design. The main motivation for our work is the observation that *current virtual switch architectures are not well-suited for multi-tenancy*. This observation leads us to revisit the fundamental design of secure vswitches. Hence, in this paper, we present, implement, and evaluate a multi-tenant (virtual) switch architecture, *MTS*, which extends the benefits of multi-tenancy to the vswitch in a secure manner, without imposing prohibitive resource requirements or jeopardizing performance.

Fig. 1 illustrates the key idea underlying the *MTS* design, by showing the security-performance-resource tradeoffs for different architectures. The current vswitch architecture is shown in Fig. 1(a), whereby per-tenant logical datapaths share a common (physical or software) switch component deployed at the Host hypervisor layer (in the rest of this paper, we shall sometimes refer to this design point as the "Baseline"). As we argued above, this design is fundamentally insecure [66, 15] as it violates basic secure design principles, like least privilege, complete mediation, or the least common mechanism. In *MTS*, we address the least privilege principle by the *compartmentalization* of the vswitches (Fig. 1(b)): by moving the vswitches into a dedicated vswitch VM, we can prevent an attacker from compromising the Host via the vswitch [33]. Then, we establish a secure communication channel between the tenant VMs and the vswitch VM via a trusted hardware technique, Single Root Input/Output Virtualization, or SR-IOV, a common feature implemented in most modern NICs and motherboards [4, 34]. Thus, all tenant-to-tenant and tenant-Host networking is *completely mediated* via the SR-IOV NIC. Adopting Google's *extra security layer* design principle [1] which requires that between any untrusted and trusted component, there have to be at least two distinct security boundaries [7, 22], we introduce a second level of isolation by moving the vswitch, deployed into the vswitch VM, to the user space. Hence, at least two independent security mechanisms need to fail (user-kernel separation and VM-isolation) for the untrusted tenant code to gain access to the Host.

Interestingly, we are able to show the resultant secure vswitch design, which we call the *single vswitch VM* design, does not come at the cost of performance; just the contrary, our evaluations show that we can considerably improve throughput and latency, for a relatively small price in resources. Finally, we introduce a "hardened" *MTS* design that we call the *multiple vswitch VMs* design (Fig. 1(c)), whereby, in line with the principle of the least common mechanism, we further separate the vswitch by creating multiple separate vswitch VMs, one for each tenant (or based on security zones/classes). This way, we can maintain full network isolation for multiple tenants.

Contributions. Our main contributions in this paper are:

- We *identify requirements and design principles* that can prevent the virtual switch from being a liability to virtualization in the cloud, and we carefully apply these principles to revisit multi-tenancy in virtual networking.

- We *present* MTS*, a secure vswitch design* whereby the vswitch is moved into a separate VM that prevents malicious tenants from compromising the Host via the virtual switch, and we also show a "hardened" *MTS* design that also prevents compromising *other* tenants' virtual

networks through the vswitch. All our designs are incrementally deployable, providing an inexpensive deployment experience for cloud operators.

- We *report on extensive experiments* with our *MTS* prototype and we find a noteworthy improvement (1.5-2x) in throughput compared to the Baseline, with similar or better latency for an extra CPU. We build our prototype from off-the-shelf commodity components and existing software; *MTS* and the data from this paper are available at:

  `https://www.github.com/securedataplane`

Organization. We dive deeper into designing a secure vswitch in Section 2. In Section 3 we elaborate on *MTS* and report on two evaluations in Sections 4 and 5. We enter a discussion of *MTS* in Section 6, review related work in Section 7 and finally draw conclusions in Section 8.

2 Securing Virtual Switches

As demonstrated in previous work [15, 66], the current state-of-the-art in virtual switch design can be exploited to not only break network isolation, but also to break out of a virtual machine. This motivates us to identify requirements and design principles that make virtual switches a dependable component of the data center [65].

2.1 State-of-the-Art

Virtual networks in cloud systems using virtual switches typically follow a *monolithic* architecture, where a single controller programs a *single vswitch* running in the Host OS with per-tenant logical datapaths in the vswitch. Isolation between tenants is at the level of flow-tables [37, 32, 52]: the controller populates the flow tables in each per-tenant logical datapath with sufficient flow rules to connect the tenant's Host-based VMs to the rest of the data center and the public Internet. Those sets of flow rules are complex: with a small error in one rule potentially having security consequences, e.g., making intra-tenant traffic visible to other tenants.

As shown in Table 1, nearly all vswitches are *monolithic* in nature. A single vswitch is installed with flow rules for all the tenants hosted on the respective server. This increases the trusted computing base (TCB) of the single vswitch, as it is responsible for Layer 2-7 of the virtual networking stack. Next, nearly 80% of the surveyed vswitches are co-located with the Host virtualization layer. This increases the TCB of the server since a vswitch is a complex piece of software, consisting of tens of thousands of lines of code. The complexity of network virtualization is further increased by the fact that packet processing for roughly 70% of the virtual switches is spread across user space and the kernel (see last two columns in Table 1). These concerns are partially addressed by the current industry trend towards offloading vswitches to *smart NICs* [34, 50, 26, 40]. Indeed consolidating the vswitch into the NIC can improve the security as it reduces the TCB of the Host. These burgeoning architectures, however, share the main trait that the per-tenant logical datapaths are monolithic, often with full privilege and direct access to the Host OS, which when compromised can break network isolation and be used as a stepping-stone to the Host.

Table 1: Design characteristics of virtual switches.

Name	Ref.	Year	Emphasis	Monolithic	Co-Location	Kernel	User
OvS	[53]	2009	Flexibility	✓	✓	✓	✓
Cisco NexusV	[68]	2009	Flexibility	✓	✓	✓	✗
VMware vSwitch	[69]	2009	Centralized control	✓	✓	✓	✗
Vale	[57]	2012	Performance	✓	✓	✓	✗
Research prototype	[33]	2012	Isolation	✓	✗	✓	✓
Hyper-Switch	[55]	2013	Performance	✓	✓	✓	✓
MS HyperV-Switch	[42]	2013	Centralized control	✓	✓	✓	✗
NetVM	[28]	2014	Performance, NFV	✓	✓	✗	✓
sv3	[63]	2014	Security	✗	✓	✗	✓
fd.io	[64]	2015	Performance	✓	✓	✗	✓
mSwitch	[27]	2015	Performance	✓	✓	✓	✗
BESS	[8]	2015	Programmability, NFV	✓	✓	✗	✓
PISCES	[61]	2016	Programmability	✓	✓	✓	✓
OvS with DPDK	[58]	2016	Performance	✓	✓	✗	✓
ESwitch	[44]	2016	Performance	✓	✓	✗	✓
MS VFP	[20]	2017	Performance, flexibility	✓	✓	✓	✗
Mellanox BlueField	[40]	2017	CPU offload	✓	✗	✓	✓
Liquid IO	[50]	2017	CPU offload	✓	✗	✓	✓
Stingray	[26]	2017	CPU offload	✓	✗	✓	✓
GPU-based OvS	[67]	2017	Acceleration	✓	✓	✓	✓
MS AccelNet	[21]	2018	Performance, flexibility	✓	✓	✓	✗
Google Andromeda	[16]	2018	Flexibility and performance	✓	✓	✗	✓

2.2 Threat Model

We assume the attacker's goal is to either escape network virtualization by compromising the virtual switch, or to tamper with other tenant's network traffic by controlling the virtual switch [66]. Hence, she can affordably rent a VM in an Infrastructure-as-a-Service (IaaS) cloud, or has somehow managed to compromise a VM, e.g., by exploiting a web-server vulnerability [14]. From the VM she can send arbitrary packets, make arbitrary computations, and store arbitrary data. However, she does not have direct control to configure the Host OS and hardware: all configuration access happens through a dedicated cloud management system.

The defender is a public cloud provider who wants to prevent the attacker from compromising virtual network isolation; in particular, *the cloud provider wants to maintain tenant-isolation even when the vswitch is compromised*. We assume that the cloud provider already supports SR-IOV at NICs [4, 34, 21] and the underlying virtualization and net-

work infrastructure is trusted, including the hypervisor layer, NICs, firmware, drivers, core switches, and so on.

2.3 Design Principles and Security Levels

Our *MTS* design is based on the application of the secure system design principles, established by Saltzer et al. [59] (see also Bishop [9] and Colp et al. [12]), to the problem space of virtual switches.

Least privilege vswitch. The vswitch should have the minimal privileges sufficient to complete its task, which is to process packets to and from the tenant VMs. Doing so limits the damage that can result from a system compromise or misconfiguration. Current best-practice is, however, to run the vswitch co-located with the Host OS and with elevated privileges; prior work has shown the types and severity of attacks that can happen when this principle fails [66]. A well-known means to the principle of least privilege is *compartmentalization*: execute the vswitch in an isolated environment with limited privileges and minimal access to the rest of the system. In the next section, we will show how *MTS* implements compartmentalization by committing the vswitches into one or more dedicated vswitch VMs.

Complete mediation of tenant-to-tenant and tenant-to-host networking. This principle requires that the network communication between the untrusted tenants and the trusted Host is completely mediated by a trusted intermediary to prevent undesired communication. This principle, when systematically applied, may go a long way towards reducing the vswitch attack surface. By *channeling all network communication* between untrusted and trusted components *via a trusted intermediary* (a so called reference monitor), the communication can be validated, monitored and logged based on security policies. In the next section, we show how complete mediation is realized in *MTS* using a secure SR-IOV channel between the tenant VMs, vswitches and Host.

Extra security boundary between the tenant and the host. This security principle, widely deployed at Google [1], requires that between any untrusted and trusted component there has to be at least two distinct security boundaries, so at least two independent security mechanisms need to fail for the untrusted component to gain access to the sensitive component [7]. We establish this extra layer of security in *MTS* by *moving the vswitch to user space*. This also contributes to implementing the "least privilege" principle: the user-space vswitch can drop administrator privileges after initialization.

Least common mechanisms. This principle addresses the amount of infrastructure shared between tenants; applied to the context of vswitches this principle requires that the network resources (code paths, configuration, caches) common to more than one tenant should be minimized. Indeed, every shared resource may become a covert channel [9]. Correspondingly, *decomposing the vswitches themselves into multiple compartments* could lead to hardened vswitch designs.

Security levels. From these principles, we can obtain different levels of security:

- **Baseline:** The per-tenant logical datapaths are consolidated into a single physical or software vswitch that is co-located with the Host OS.
- **Level-1:** Placing the vswitch in a dedicated compartment provides a first level of security by protecting from malicious tenants to compromise the Host OS via the vswitch ("single vswitch VM" in Fig. 1b).
- **Level-2:** Splitting the vswitches into multiple compartments (based on security zones or on a per-tenant basis) adds another level of security, by isolating tenants' vswitches from each other ("multiple vswitch VMs" in Fig. 1c).
- **Level-3:** Moving the vswitches into user space, combined with Baseline or Level-1 or -2, reduces the impact of a compromise and further reduces the attack surface.

3 The *MTS* Architecture

We designed *MTS* with secure design principles from Section 2.3. We first provide an overview and then present our architecture in detail.

3.1 Overview

Compartmentalization. There are many ways in which isolated vswitch compartments can be implemented: full-blown VMs, OS-level sandboxes (jails, zones, containers, plain-old user-space processes [22], and exotic combinations of these [35, 25]), hardware-supported enclaves (Intel's SGX) [48, 3], or even safe programming language compilers (Rust), runtimes (eBPF), and instruction sets (Intel MPX). For flexibility, simplicity, and ease of deployment, *MTS relies on conventional VMs as the main unit of compartmentalization*.

VMs provide a rather strong level of isolation and are widely supported in hardware, software, and management systems. This in no way means that VM-based vswitches are mandatory for *MTS*, just that this approach offers the highest flexibility for prototyping. For simplicity, Fig. 2 depicts two vswitch compartments (Red and Blue solid boxes) running independent vswitches in their isolated VMs. The multiple compartments further reduce the common mechanisms between the vswitch and the connected tenants, achieving security Level-2. Security Level-1, although not depicted, would involve only a single vswitch VM.

Complete mediation. To mediate all interactions between untrusted tenant code and the Host OS through the vswitch, we need a secure and high-performance communication medium between the corresponding compartments/VMs. In

MTS we use Single Root IO Virtualization (SR-IOV) to interconnect the vswitch compartments (see Figure 2).

SR-IOV is a PCI-SIG standard to make a single PCIe device, e.g., a NIC, appear as multiple PCIe devices that can then be attached to multiple VMs. An SR-IOV device has one or more physical functions (PFs) and one or more virtual functions (VFs), where the PFs are typically attached to the Host and the VFs to the VMs. Only the Host OS driver has privileges to configure the PFs and VFs. The NIC driver in the VMs in turn have restricted access to VF configuration. Only via the Host, VFs and PFs can be configured with unique MAC addresses and Vlan tags. Network communication between the PFs and VFs occurs via an L2 switch implemented in the NIC based on the IEEE Virtual Ethernet Bridging standard [36]. This enables Ethernet communication not only from and to the respective VMs (vswitch and tenants) based on the destination VF's MAC address but also to the external networks.

Sharing the NIC SR-IOV VF driver and the Layer 2 network virtualization mechanism implemented by the SR-IOV NIC is considerably simpler than including the NIC driver and the entire network virtualization stack (Layer 2-7) in the TCB. Tenants already share SR-IOV NIC drivers in public clouds [4, 34, 5]. Virtual networks can be built as we will see next, as per-tenant user-space applications implementing Layer 3-7 of the virtual networking stack.

Thanks to the use of SR-IOV in *MTS*, packets to and from tenant VMs completely bypass the Host OS; instead, all potentially malicious traffic is channeled through the trusted hardware medium (SR-IOV NIC) to the vswitch VM(s). Furthermore, using SR-IOV reduces CPU overhead and improves performance (see Section 4). Finally, SR-IOV provides an attractive upgrade path towards fully offloaded, smart-NIC based virtual networking: chip [**?**] and OS vendors [**?**, **?**] have been supporting SR-IOV for many years now at a reasonable price, major cloud providers already have SR-IOV NICs deployed in their data centers [4, 34, 5], and, perhaps most importantly, this design choice liberates us from having to re-implement low-level and complex network components [33]: we can simply use any desired vswitch, deploy it into a vswitch VM, configure and attach VFs to route tenants' traffic through the vswitch, and start processing packets right away.

User-space packet processing. As discussed previously, we may choose to deploy the vswitches into the vswitch VM user-space to establish an extra security boundary between the tenant and the Host OS (Level-3 design). Thanks to the advances in kernel bypass techniques, several high-performance and feature-rich user-space packet processing frameworks are available today, such as Netmap [56], FD.IO [64], or Intel's DPDK [29]. Our current design of *MTS leverages OvS with the DPDK datapath for implementing the vswitches* [58]. DPDK is widely supported, it has already been integrated with popular virtual switch products,

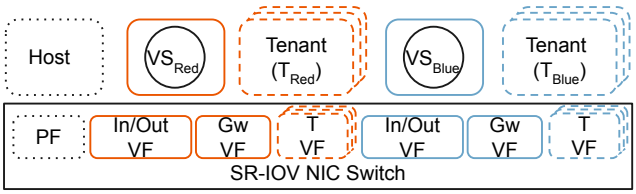

Figure 2: High-level overview of *MTS* in security Level-2. The Red and Blue vswitch compartments (VMs) are allocated dedicated virtual functions (VFs) to communicate with external networks using the *In/Out VF*, their respective tenants using the *Gw VF* and *T VF*. Communication between the vswitches, tenants and the Host physical function (PF) are mediated via the SR-IOV NIC switch.

and extensive operational experience is available regarding the expected performance and resource footprint [38]. Note, however, that using DPDK and OvS is not mandatory in *MTS*; in fact, thanks to the flexibility provided by our VM-compartments and SR-IOV, we can deploy essentially any user-space vswitch to support *MTS*.

3.2 Detailed Architecture

For the below discussion, we consider the operation of *MTS* for one vswitch compartment and its corresponding tenant VMs from the Level-2 design shown in Fig. 2. The case when only a single compartment (Level-1) is used is similar in vein: the flow table entries installed into the vswitch and the VFs attached to the vswitch compartment need to be modified somewhat; for lack of space we do not detail the Level-1 design any further.

Connectivity. Each vswitch VM is allocated at least one VF (In/Out VF) for external (inter-server) connectivity and another as a gateway (Gw VF) for vswitch-VM-to-tenant-VM connectivity as shown in Fig. 2. Isolation between the external and the tenant network (tenant VF shown as T VF in the Figure) is enforced at the NIC-level by configuring the Gw VF and the tenant VFs with a Vlan tag specific to the tenant. Different Vlan tags are used to further isolate the multiple vswitch compartments and their resp. tenants on that server.

The packets between VFs/PFs in the NIC are forwarded based on the destination MAC address and securely isolated using Vlan tags (the same security model as provided by enterprise Ethernet switches). For all packets to and from the tenant VMs to pass through the vswitch-VM, the destination MAC address of each packet entering and leaving the NIC needs to be accurately set, otherwise packets will not reach the correct destination. This can be addressed by introducing minor configuration changes to the normal operation of the tenant and the vswitches, detailed below.

Ingress chain. Fig. 3ⓐ illustrates the process by which packets from an external network reach the tenant VMs. In step ① a packet enters the server through the NIC fabric

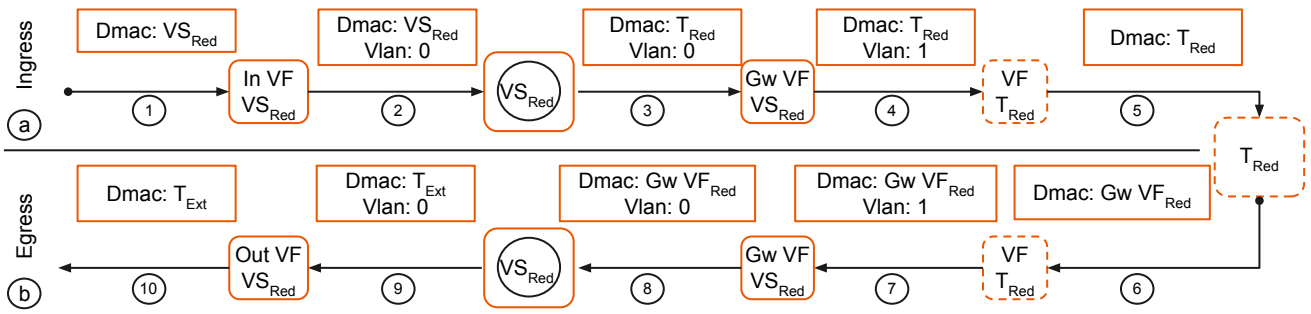

Figure 3: A step-by-step illustration of how packets enter and leave the Red tenant from Figure 2 in *MTS*. ⓐ shows how ingress packets reach Tenant$_{Red}$. ⓑ shows how Tenant$_{Red}$ packets reach an external system Tenant$_{Ext}$.

port having the Red vswitch's In/Out VF MAC address as the destination MAC address (Dmac). The NIC switch will deliver the packet to the vswitch VM untagged (Vlan 0) in ②. The Red vswitch then uses the destination IP address in the packet to identify the correct tenant VM to send the packet to, changes the destination MAC address to that of the Red tenant's VF (VF T_{Red}), and emits the packet to the Gw VF in the NIC in ③. This ensures accurate packet delivery to and from tenant VMs and the complete isolation of the tenant-vswitch traffic from other traffic instances. In ④ and ⑤, the NIC tags the packet with the Red tenant's specific Vlan tag (Vlan 1 in the figure), uses the built-in switch functionality to make a lookup in the MAC learning table for the Vlan, pops the Vlan tag and finally forwards the packet to the Red tenant's VM. The NIC forwarding process is completely transparent to the vswitch and tenant VMs, the only downside is the extra round-trip to the NIC. Later we show that this round-trip introduces negligible latency overhead.

Egress chain. The reverse direction shown in Fig. 3ⓑ, sending a packet from the tenant VM through the vswitch to the external network goes in similar vein. In ⑥ the Red tenant VM T_{Red} sends a packet through its VF (T_{Red}) with the destination MAC address set to the MAC address of the Red tenant's Gw VF; in the next subsection we describe two ways to achieve this. In ⑦ the NIC switch tags the packet (Vlan 1), looks-up the destination MAC address which results in sending the packet to the Gw VF. At the gateway VF ⑧, the NIC switch pops the Vlan tag and delivers the packet to the Red vswitch VM. The vswitch receives the packet, looks up the destination IP address, rewrites the MAC address to the actual (external) gateway's MAC address, and then sends the packet out to the In/Out VF in ⑨. Finally in ⑩, the NIC will in turn send the packet out the physical fabric port.

Communication between the two VMs of a single tenant inside the server goes similarly, with the additional complexity that packets now take *two* extra round-trips to the NIC: once on the way from the sender VM to vswitch, and once on the way from the vswitch to the destination VM. Again, our evaluations in the next sections will show that the induced latency overhead for such a traffic scenario is low.

System support. Next, we detail the modifications the cloud operator needs to apply to the conventional vswitch setup to support *MTS*. The primary requirement is to modify the centralized controllers to appropriately configure tenant specific VFs with Vlan tags and MAC addresses, and insert correct flow rules to ensure the vswitch-tenant connectivity. Second, advanced multi-tenant cloud systems rely on tunneling protocols to support L2 virtual networks. This is also supported by *MTS*, by modifying the flow tables to pop/insert the appropriate headers whenever packets need to be decapsulated/encapsulated. Note that after decapsulation the tunnel id can be used in conjunction with the destination IP address to identify the appropriate tenant VM. Third, the ARP entry for the default gateway must be appropriately set in each tenant VM so that packets from the tenant VM go to the vswitch VM. To this end, the tenant VMs can be configured with a static ARP entry pointing to the appropriate Gw VF, or using the centralized controller and vswitch as a proxy-ARP/ARP-responder [45]. Finally, to prevent malicious tenants from launching an attack on the system, the cloud operator needs to deploy security filters in the NIC. In particular, source MAC address spoofing prevention must be enabled on all tenant VMs' VFs. Furthermore, flow-based wildcard filters can also be applied in the NIC for additional security, e.g., to drop packets not destined to the vswitch compartment, to prevent the Host from receiving packets from the tenant VMs, etc. Our *MTS* implementation, described in Section 4, takes care of removing the manual management burden in applying the above steps.

Resource allocation. Additional levels of security usually come with increased resource requirement, needed to run the security/isolation infrastructure. Below, we describe two resource sharing strategies and how the VFs are allocated to the vswitch compartments. However, due to the sheer quantity and diversity in cloud setups, we restrict the discussion to plain compute and memory resources and the number of SR-IOV VFs for the different *MTS* security levels.

We consider two modes for compute and memory resources. A *shared* mode where tenants' vswitches share a single physical CPU core, while in the *isolated* mode each tenant's vswitch is pinned to a different core. However, we

assume that each vswitch compartment gets an equal share of main memory (ram) and this is inexpensive compared to physical CPU cores. Dedicating compute and memory resources for vswitching is not uncommon among cloud operators [21, 16]. Note that the *shared* and *isolated* resource allocations are merely two ends of the resource allocation spectrum, different sets of vswitch VMs could be allocated resources differently, e.g., based on application or customer requirements. In the next section we will see that the resource requirement for multiple vswitch VM compartments, i.e., Level-2 alone, is not resource prohibitive in the *shared* mode, however, Level-2 and Level-3 can be.

Regarding the number of SR-IOV VFs needed, the current standard allows each SR-IOV device to have up to 64 VFs per PF. For Level-1, the total number of VFs is given by the sum of i) the number of VFs allocated for external connectivity (In/Out VF); ii) the total number of tenant-specific gateway VFs; and iii) tenant-specific VM VFs hosted on the server. In a basic Level-1 setup hosting 1 tenant, with 1 In/Out VF and 1 gateway VF and 1 VF for the tenant VM, the total VFs is 3. Similarly for 4 tenants, the total VFs is 9. For Level-2, the total number of VFs is given by the sum of i) the tenant-specific VFs allocated for external connectivity; ii) the tenant-specific gateway VFs; and iii) tenant-specific VM VFs hosted on the server. For a basic Level-2 setup hosting 2 tenants, with 1 In/Out VF, 1 gateway VF per tenant vswitch and 1 VF for each tenant VM, the total VFs is 6. Similarly for 4 tenants, the total VFs is 12.

4 Evaluating Tradeoffs

We designed a set of experiments to empirically evaluate the security-performance-resource tradeoff of *MTS*. To this end, we measure *MTS*'s performance for different security levels under different resource allocation strategies, in canonical cloud traffic scenarios [18]. The focus is on throughput and latency performance metrics, and physical cores and memory for resources. In particular, the experiments serve to verify our expectation that our design does not introduce a considerable overhead in performance. However, we do expect the amount of resources consumed to increase; our aim is to quantify this increase in different realistic setups.

Prototype framework. We took a programmatic approach to our design and evaluation, hence, we developed a set of primitives that can be composed to configure *MTS* to conduct all the experiments described in this paper. Hence, as a first step we do not consider complex cloud management systems (CMS) such as OpenStack; this way we can conduct self-contained experiments without the possible interference cause by a CMS. Our framework is written in Python and currently supports OvS and ovs-DPDK as the base virtual switch, Mellanox NIC, and the `libvirt` virtualization framework. Our framework and data are available on-line at the following URL: `https://www.github.com/securedataplane`

Methodology. We chose a set of standard cloud traffic scenarios (see Fig. 4) and a fixed number of tenants (4). For each of those scenarios, we allocated the necessary resources (Sec. 3) and then configured the vswitch either in its default configuration (Baseline) or one of the three security levels (Sec. 2.3). The system was then connected in a measurement setup to measure the one-way forwarding performance. Important details on the topology, resources, security levels and the hardware and software used are described next.

Traffic scenarios. The three scenarios evaluated are shown in Fig. 4. *Physical-to-physical (p2p):* Packets are forwarded by the vswitch from the ingress physical port to the egress. This is meant to shed light on basic vswitch forwarding performance. *Physical-to-virtual (p2v):* Packets are forwarded by the vswitch from one physical port to a tenant VM, and then back from the tenant VM to the other physical port. Compared to the p2p, this will show the overhead to forward to and from the tenant VM. *Virtual-to-virtual (v2v):* Similar to the p2v, however, when the packets return from the tenant to the vswitch, the vswitch sends the packet to another tenant which then sends it back to the vswitch and then out the egress port. This scenario emulates service chains in network function virtualization. Since the path length increases from p2p to p2v to v2v, we expect the latency to increase and the throughput to decrease when going from p2p to p2v to v2v.

Resources. We allocated compute resources in the following two ways. *Shared:* All vswitch compartments share 1 physical CPU core and their associated cache levels. *Isolated:* Each vswitch compartment is allocated 1 physical CPU core and their associated cache levels. In case of the Baseline, we allocated cores proportional to the number of vswitch compartments, e.g., 2 cores to compare with 2 vswitch VMs. For main memory, each VM (vswitch and tenant) was allocated 4 GB of which 1 GB is reserved as one 1GB Huge page. Similarly, for the Baseline, a proportional amount of Huge pages was allocated. When using *MTS*, each vswitch VM was allocated 2 In/Out VFs (1 per physical port), and 2 appropriately Vlan tagged Gw VFs per tenant (1 per physical port). When DPDK was used in Level-3: one physical core needs to allocated for each ovs-DPDK compartment (including the Baseline), hence, only the isolated mode was used; all In/Out, gateway and tenant ports connected to OvS were assigned DPDK ports (in the case of the Baseline, the tenant port type was the dpdkvhostuserclient [17]). All the tenant VMs got two physical cores and two VFs, 1 per port (these are VMs the tenant would use to run her application) so that the forwarding app ($l2fwd$) could run without being a bottleneck.

Security levels and tenants. For each resource allocation mode, we configured our setup either in Baseline or one of the three *MTS* security levels (Section 2.3). In the Baseline and Level-1, there were 4 tenant VMs connected to the

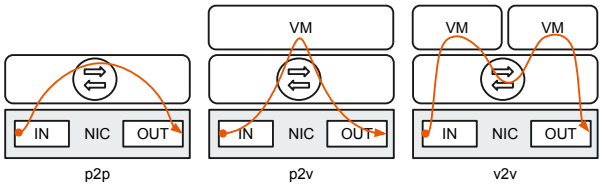

Figure 4: Traffic scenarios evaluated.

vswitch. For Level-2, we configured 2 vswitch VMs and each vswitch had 2 tenant VMs, and then we configured 4 vswitch VMs where each vswitch VM had 1 tenant VM. We repeated Level-3 with Baseline, Level-1 and the two Level-2 configurations.

Setup. To accurately measure the one-way forwarding performance (throughput and latency), we used two servers connected to each other via 10G short range optical links. The device under test (DUT) server was an Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz with 64 GB of RAM with the IOMMU enabled but hyper-threading and energy efficiency disabled, and a 2x10G Mellanox ConnectX4-LN NIC with adaptive interrupt moderation and irq balancing disabled. The other server was the packet/load generator (LG), sink and monitor, with an Endace Dag 10X4-P card (which gives us accurate and precise hardware timestamps) [19]. The link between the LG and DUT, and DUT and sink were monitored via a passive optical network tap connected to the Dag card. Each receive stream of the Dag card was allocated 4 GB to receive packets. The Host, vswitch VM and tenant VMs used the Linux kernel 4.4.0-116-generic, Mellanox OFED linux driver 4.3-1.0.1.0, OvS-2.9.0 and DPDK 17.11. Libvirt 1.3.1 was used with QEMU 2.5.0. In the tenant VMs, we adapted the DPDK-17.11 $l2fwd$ app to rewrite the correct destination MAC address when using *MTS*, and used the default $l2fwd$ drain-interval (100 microseconds) and burst size (32) parameters. For the Baseline, we used the default linux bridge in the tenant VMs as using DPDK in the tenant without being backed by QEMU and OvS (e.g., dpdkvhostuserclient) is not a recommended configuration [39]. For network performance measurements, we used Endace dag-5.6.0 software tools (dagflood, dagbits, and dagsnap).

4.1 Throughput

Our first performance tradeoff is evaluating the forwarding throughput. This will shed light on the packets per second (pps) processing performance of *MTS* compared to the Baseline. It also uncovers packet loss sooner than measuring the bandwidth [31]. We measure the aggregate throughput with a constant stream of 64 B packets replayed at line rate (14 Mpps) by the LG and collected at the sink. Since we fixed the number of tenants to 4, the stream of packets comprises 4 flows, each to a respective tenant VM identified by the destination MAC and IP address. At the monitor we collect the packets forwarded to report the aggregate throughput. Each experimental run lasts for 110 seconds and measurements are made from the 10-100 second marks.

Results. The throughput measurement data for the shared mode is shown in Fig. 5(a). In Fig. 5(d) we can see the data for the isolated mode and in Fig. 5(g) the data for Level-3 in the isolated mode is shown. From Figures 5(a) and (d) we can see that nearly always MTS *had either the same or higher aggregate throughput than the Baseline*. The improvement in throughput is most obvious in the p2v and v2v topologies as vswitch-to-tenant communication is via the PCIe bus and NIC switch, which turns out to be faster than Baseline's memory bus and software approach. Sharing the physical core for multiple compartments (Fig. 5(a)) in the p2v and v2v scenarios can offer 4x isolation (Level-2 with 4 compartments) and a 2x increase in throughput (nearly .4 Mpps and .2 Mpps) compared to the Baseline (nearly .2 Mpps and .1 Mpps).

Fig. 5(d) is noteworthy as multiple cores for vswitch VMs and the Baseline functions as a load-balancer when isolating the CPU cores. In the p2p scenario, the aggregate throughput increases roughly from 1 Mpps to 2 Mpps to 4 Mpps as the number of cores increase. We observe that *MTS* is slightly more than the Baseline in the p2p, however, in the p2v and v2v scenarios *MTS* offers higher aggregate throughput. As expected, using DPDK can offer an order of magnitude better throughput (Fig. 5(g)). In the p2p topology, we were able to nearly reach line rate (14.4 Mpps) with four DPDK compartments as the packets were load-balanced across the multiple vswitch VMs, while the Baseline was able to saturate the link with 2 cores. With *MTS*, the throughput saturates (at around 2.3 Mpps) in the p2v and v2v topologies because several ports are polled using a single core and packets have to bounce off the NIC twice as much compared to the Baseline where we observe nearly twice the throughput for 2 and 4 cores. Nevertheless, we can see a slight increase in the throughput of *MTS* as the vswitch VMs increase, because the number of ports per vswitch VM decreases as the number of vswitch VMs increase. Due to the limited physical cores on the DUT, we could not evaluate 4 vswitch VMs in the v2v topology as it required more cores and ram than available.

Key findings. The key result here is that *MTS* offers increasing levels of security with comparable, if not increasing levels of throughput in the shared and isolated resource modes, however, the Baseline's throughput with user-space packet processing (DPDK) is better than *MTS*.

4.2 Latency

The second performance tradeoff we evaluated was the forwarding latency, in particular, we studied the impact of packet size on forwarding. We selected 64B (minimum IPv4 UDP packet size), 512B (average packet), 1500B (maximum MTU) packets and 2048B packets (small jumbo frame). As

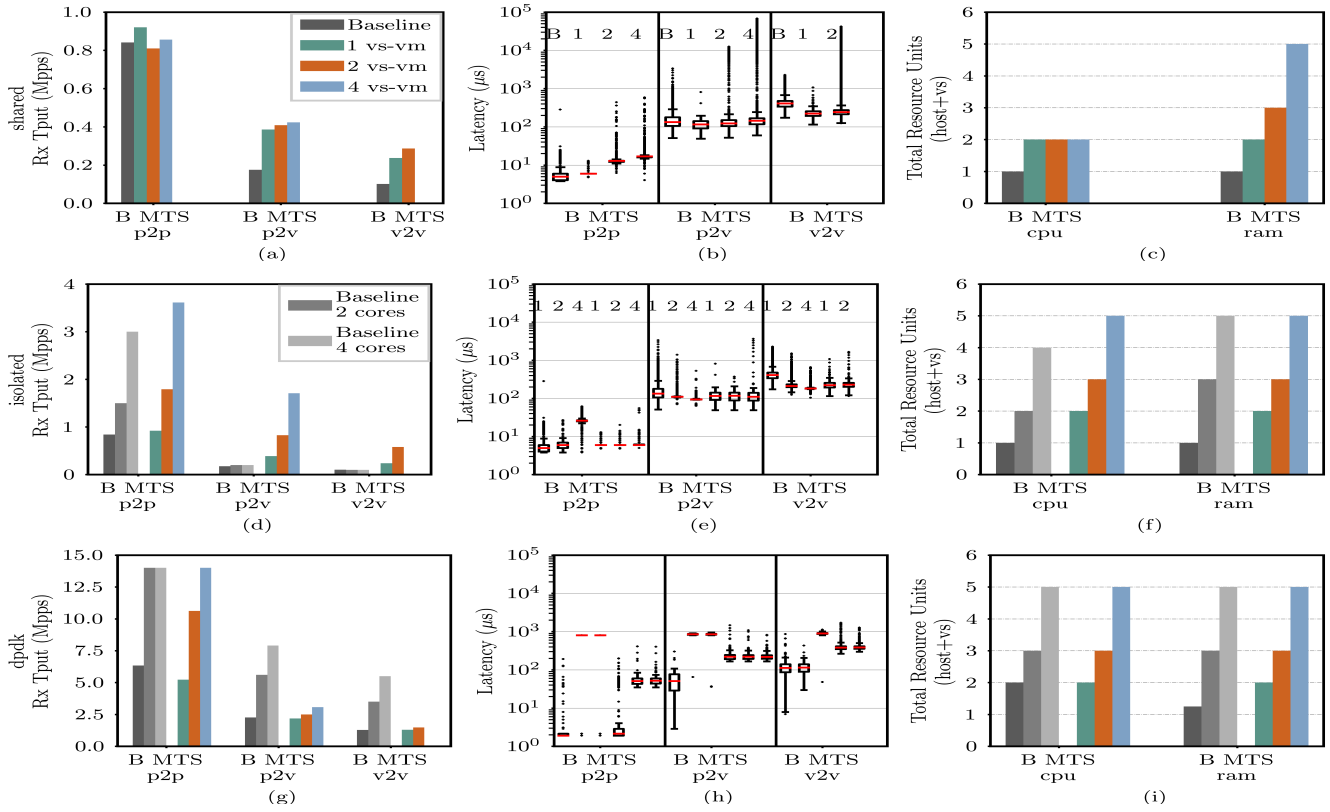

Figure 5: The security, throughput, latency and resource tradeoff comparison of *MTS*. The rows indicate the resource mode. The columns are ordered as throughput, latency and resources. The security levels used are shown in the legend. Note the bottom row is for security Level-3 in the isolated resource mode combined with other security levels.

in the throughput experiments, we used 4 flows, one to each tenant. For each experimental run, we continuously sent packets from the LG to the sink via the DUT at 10 kpps for 30 seconds. Note that is the aggregate throughput sent to the NIC and not to the vswitch VM. To eliminate possible warm-up effects, we only evaluated the packets from the 10-20 second mark.

Results. For brevity the latency distribution only for 64 *B* packets is reported here. Fig. 5(b) shows the data for the shared mode, while Fig. 5(e) is for the isolated mode. Level-3 latency data is shown in Fig. 5(h). Although the p2p scenarios shows that *MTS* increases the latency (Fig. 5(b), (e) and (h)), the p2v and v2v scenarios show that *MTS* is slightly faster than the Baseline. This is for two reasons. First, packets between the vswitch and the tenant VMs pass through the SR-IOV NIC (PCIe bus) rather than a software only vswitch (memory bus). Second, when using the Baseline the tenant uses the Linux bridge. The exception to this can be seen with user-space packet processing (Fig. 5(h)), where the Baseline with a single core for dpdk (2 in total) is always faster than *MTS*. As mentioned in Section. 4.1, due to resource limitations we could not evaluate the 4 vswitch VMs in v2v.

The variance in latency increases as more compartments share the same physical core (Fig. 5(b)). Isolating the vswitch VM cores leads to more predictable latency as seen in Fig. 5(e). When using DPDK (Fig. 5(h)) we make two observations: i) *MTS* takes longer to forward packets than without using DPDK; ii) the latency for Baseline with 2 and 4 cores for dpdk (3 and 5 in total) is unexpectedly high (around 1 ms). Regarding the former, we conclude that *MTS* with OvS and DPDK requires further tuning as we used the default OvS-DPDK parameters for the drain interval, batch size and huge pages: There is an inherent tradeoff between high throughput and average per-packet latency when using a shared memory model where a core is constantly polling [**?**]. For the latter, we observe that the throughput of 10 kpps is too low to drain the multiple queues on the DPDK ports. At 100 kpps and 1 Mpps, we measured an approximately 2 microsecond latency for the p2p scenario.

Key findings. We observe that for the shared mode, and 4x compartmentalization (Level-2), the latency is comparable to the Baseline (p2v) with a lot of variance whereas when isolated the latency is more predictable.

4.3 Resources

In Fig. 5(c), (f) and (i) we see the total CPU and memory consumption for Baseline and *MTS*. Note that across all the

figures, one core and at least one Huge page is always dedicated for the Host OS. In the case of the (single core) Baseline, the vswitch (OvS) runs in the Host OS and hence shares the Host's core and ram. However, for the single vswitch VM in the shared, isolated and DPDK modes, the Host OS consumes one core and the vswitch VM consumes another core making the total CPU cores two. Similarly, the 2 and 4 vswitch VMs in the shared mode, also consume the same number of cores as the single vswitch VM but a linear increase in ram. In the isolated mode, *MTS* consumes only one extra physical core relative to the Baseline, and in DPDK, *MTS* and Baseline consume equal number of cores. With respect to the memory consumption, we note that *MTS*'s and Baseline's memory consumption in the isolated and DPDK modes are the same.

Hence, we conclude that for one extra physical core, *MTS* offers multiple compartments, making the shared resource allocation economically attractive. The resource cost goes up when user-space packet processing is introduced or isolating cores, making it relatively expensive for multiple vswitch VMs.

Key findings. (i) High levels (2x/4x) of virtual network isolation per server can be achieved with an increase in aggregate throughput (2x) in the shared mode; (ii) for applications that require low and predictable latency, vswitch compartments should use the isolated mode; (iii) although user-space packet processing using DPDK offers high throughput, it is expensive (physical CPU and energy costs).

5 Workload-based Evaluation

We also conducted experiments with real workloads, to gain insights on how cloud applications such as web servers and key-value stores will perform as tenant applications are the end hosts of the virtual networks.

Methodology. For simplicity we focus our workload-based evaluation only on TCP applications as our previous measurements dealt with UDP. In general, we use a similar methodology to the one described in Section 4. For all the TCP-based measurements, we configured the tenant VMs to run the respective TCP server and from the client (LG) we benchmark the server to measure the throughput and/or response time. The topologies, resources and setup used to make these measurements are slightly nuanced which we highlight next.

Traffic scenarios. Only the p2v and v2v patterns are evaluated with workloads as we want to understand the performance of applications hosted in the server.

Resources. The ingress and egress ports for all the traffic are on the same physical NIC port unlike in the previous section where the ingress and egress ports were on separated physical ports of the NIC. Hence, each tenant's vswitch VM was given 1 VF for In/Out and 1 tagged Gw VF. Each tenant VM was given 1 VF.

Setup. The applications generating the load are standard TCP, Apache and Memcached benchmarking tools respectively Iperf3 v3.0.11 [30], ApacheBench v2.3 (ab) [2] and libMemcached v1.0.15 (memslap) [41]. Instead of the Endace card we used a similar Mellanox card at the LG.

5.1 Workloads and Results

Iperf: To compare the maximum achievable TCP throughput, we ran Iperf clients for 100 *s* with a single stream from the LG to the respective Iperf servers in the DUT's tenant VM. The aggregate throughput was then reported as the sum of throughput for each client-server. We collected 5 such measurements for each experimental configuration and report the mean with 95% confidence.

Webserver: To study workloads from webservers (a very common cloud application), we consider the open-source Apache web server. Using the ApacheBench tool from the LG, we benchmarked the respective tenant webservers by requesting a static 11.3 KB web page from four clients (one for each webserver). Each client made up to 1,000 concurrent connections for 100 *s* after which we collected the throughput and latency statistics reported by ApacheBench. In the v2v scenario, we used only two client-servers as one of the tenant VMs simply forwarded packets using the DPDK $l2fwd$ app. We collected 5 such repetitions to finally report the average throughput and latency for each experimental configuration with 95% confidence.

Key-value store: Key-value stores are also commonly used cloud applications (e.g., with with webservers). We opted for the open-source Memcached key-value store as it also has an open-source benchmarking tool libMemcached-memslap. We used the default Set/Get ratio of 90/10 for the measurements. The methodology and reporting of the measurements are the same as the webserver.

Results. The data for the Iperf measurements in the shared mode is shown Fig. 6(a). The data for the isolated mode is shown in Fig. 6(f) and Fig. 6(k) depicts the throughput for Level-3. As seen in Section 4.1, here too we observe that *MTS* has a higher throughput (more than 2x in the shared mode) than the Baseline except when DPDK is used in the v2v topology. *MTS* saturated the 10G link in the p2v scenario when isolated and DPDK modes were used.

The data from the throughput measurements for the Apache webserver and Memcached key-value store are first reported in the shared mode in Fig. 6(b) and (c) respectively. For the isolated mode they are shown in Fig. 6(g) and (h). Level-3 throughput is shown in Fig. 6(l) and (m). The three main results from the throughput measurements for Apache and Memcached are the following. *MTS* can offer nearly 2x throughput and 4x isolation (Level-2) in the shared mode. Apache's and Memcached's throughput saturated with *MTS*: we expected the throughput to increase as the vswitch VMs increase when the compartments have isolated cores, how-

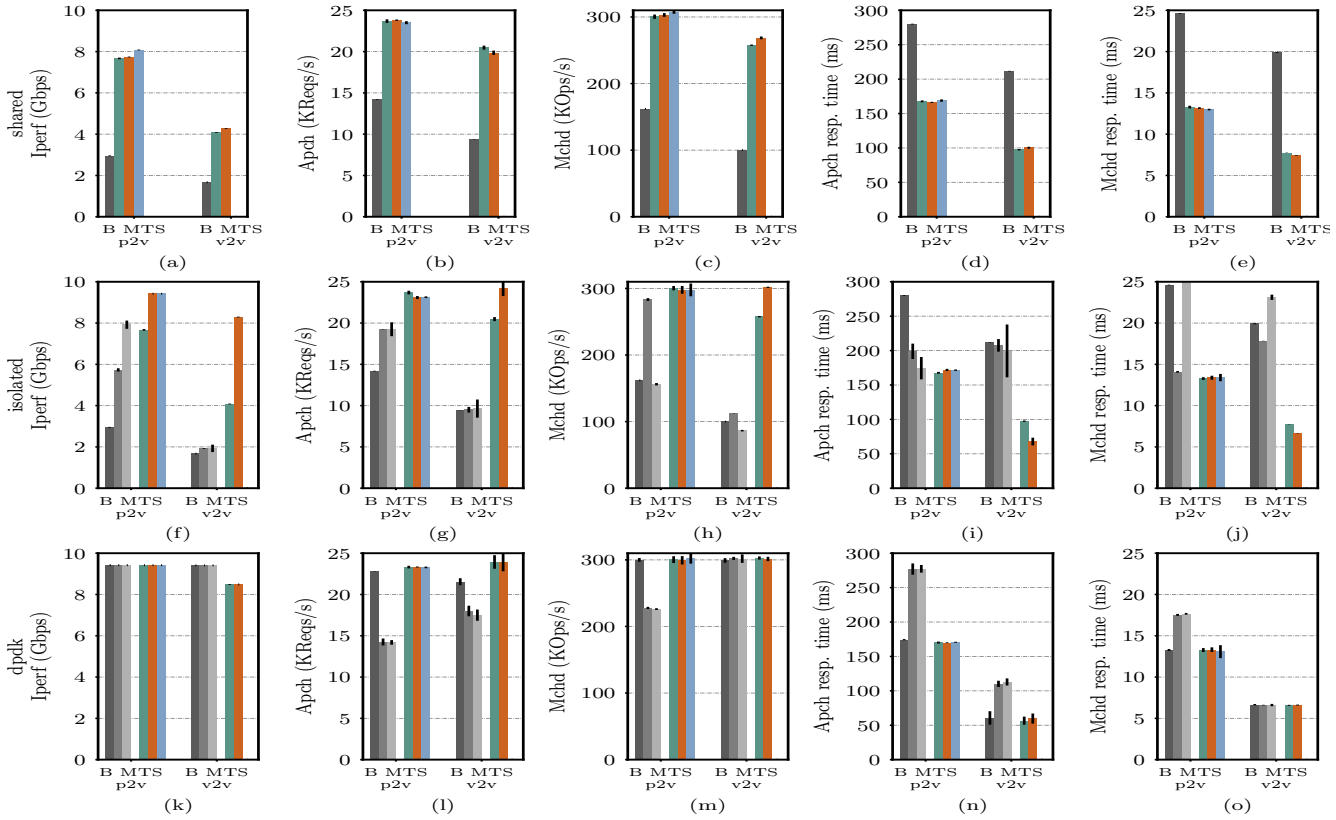

Figure 6: Iperf throughput, Apache and Memcached throughput and latency (shown in the columns) comparison of *MTS*. The rows indicate the resource mode where the bottom row is for security Level-3 in the isolated resource mode combined with the other security levels. The legend is the same as in Figure 5.

ever, we do not observe that. This is further validated when using DPDK. Apache's and Memcached's throughput are highly sensitive when the Baseline uses multiple cores in the isolated and DPDK modes which means that using 2 or more cores requires workload specific tuning to the Host: the DPDK parameters, e.g., drain interval, and the workload VMs, e.g., allocating more cores, which may not always be necessary with *MTS*.

The data from the response time measurements for the Apache webserver and Memcached key-value store are first reported in the shared mode in Fig. 6(d) and (e) respectively. For the isolated mode they are shown in Fig. 6(i) and (j). Level-3 throughput is shown in Fig. 6(n) and (o). Regarding the latency, we again discern that *MTS* can offer multiple levels of isolation and maintain a lower response time (approximately twice as fast) than the Baseline.

Key findings. Our webserver and key-value store benchmarks reveal that application throughput and latency of real application are improved by *MTS*. However, for user-space packet processing, the resource costs go up for a fractional benefit in throughput or latency. Hence, biting the bullet for shared resources, offers 4x isolation and approximately 1.5-2x application performance compared to the Baseline.

6 Discussion

Centralized control, accounting and monitoring. *MTS* introduces the possibility to realize multi-tenant virtual networks which can expose tenant/compartment specific interfaces to a logically centralized control/management plane. This opens up possibilities for full network virtualization, how to expose the interface, and also how to integrate *MTS* into existing cloud management systems in an easy and usable way. Furthermore, controllers may need to manage more device, topology and forwarding information, however, the computations (e.g., routing) should remain the same. From an accounting and billing perspective, we strongly believe that *MTS* is a new way to bill and monitor virtual networks at granularity more than a simple flow rule [23]: CPU, memory and I/O for virtual networking can be charged.

SR-IOV: a double-edged sword. If an attacker can compromise SR-IOV, she could violate isolation and in the worst case get access to the Host OS via the PF driver. Hence, a rigorous security analysis of the SR-IOV standard, implementations and SR-IOV-NIC drivers can reduce the chance of a security vulnerability. Compartmentalizing the PF driver is a promising approach [10]. Furthermore, when a vswitch VM is shared among tenants, performance isolation issues

could lead to covert channels [6] or denial-of-service attacks [62, 71]. Although not yet widely supported, VM migration with SR-IOV can be introduced [43]. SR-IOV NICs have limited VFs and MAC addresses which could limit the scaling properties of *MTS*, e.g., when using containers as compartments instead of VMs.

Evaluation limitations. The results from our experiments are from a network and application performance perspective using a 10 Gbps NIC. For a deeper understanding of the performance improvement we obtained in this paper using SR-IOV, further measurements are necessary, e.g., using the performance monitoring unit (PMU) to collect a breakdown of the packet processing latencies. Such an understanding is important and relevant when dealing with data center applications that require high NIC bandwidth, e.g., 40/100 Gbps.

As described by Neugebauer et al. [46], the PCIe bus can be a bottleneck for special data center applications (e.g., ML applications): A *typical* x8 PCIe 3.0 NIC (with a maximum payload size of 256 bytes and maximum read request of 4096 bytes) has an effective (usable) bi-directional bandwidth of approximately 50 Gbps. Hence, the usability of *MTS* with PCIe 3.0 and 8 lanes can indeed be a limitation which we did not observe in this paper. Nevertheless, increasing the lanes to x16 is one potential workaround to double the effective bandwidth to around 100 Gbps. Furthermore, with chip vendors initiating PCIe 4.0 devices [11], the PCIe bus bandwidth will increase to support intense I/O applications.

7 Related Work

There has been noteworthy research and development on isolating multi-tenant virtual networks in cloud (datacenter) networks: tunneling protocols have been standardized [70, 24], multi-tenant datacenter architectures have been proposed [37], and real cloud systems have been built by many companies [21, 16]. However, most of the previous work still co-locates the vswitch with the Host as we discussed in Section 2.1. Hence, here we discuss previous and existing attempts specifically addressing the security weakness of vswitches.

To the best of our knowledge, in 2012 Jin et al. [33] (see Research prototype in Table 1) were the first to point out the security weakness of co-locating the virtual switch with the hypervisor. However, the proposed design, while ahead of its time, (i) lacks a principled approach which this paper proposes; (ii) has only a single vswitch VM whereas *MTS* supports multiple vswitch compartments making it more robust; (iii) is resource (compute and memory) intensive as the design used shared memory between the vswitch VM and all the tenant VMs while *MTS* uses an inexpensive interrupt-based SR-IOV network card for complete mediation of tenant-vswitch-VM and tenant-host networking; (iv) requires considerable effort, expertise and tuning to integrate into virtualization system whereas *MTS* can easily be scripted into existing cloud systems.

In 2014 Stecklina [63] followed up on this work and proposed sv3, a user-space switch, which can enable multi-tenant virtual switches (see sv3 in Table 1). sv3 adopts user-space packet processing and also supports compartmentalization, i.e., the Host can run multiple vswitches. However, it is still co-located with the Host, partially adopts the security principles outlined in this paper, lacks support for *real* cloud virtual networking, and requires changes to QEMU. Our system on the other hand moves the vswitch out of the Host, supports production vswitches such as OvS and does not require any changes to QEMU.

Between 2016 and 2017, Panda et al. [49] and Neves et al. [47] took a language-centric approach to enforce data plane isolation for virtual networks. However, language-centric approaches require existing vswitches to be reprogrammed/annotated which reduces adoption. Hence the solution of using compartments and SR-IOV in *MTS* allows existing users to easily migrate using their existing software. Shahbaz et al. [61] reduced the attack surface of OvS by introducing support for the P4 domain specific language which reduces potentially vulnerable protocol parsing logic.

In 2018, Pettit et al. [51] proposed to isolate virtual switch packet processing using eBPF: which is conceptually isolating potentially vulnerable parsing code in a kernel-based runtime environment. However, the design still co-locates the virtual switch and the runtime with the Host.

8 Conclusion

This paper was motivated by the observation that while vswitches have been designed to enable multi-tenancy, today's vswitch designs lack strong isolation between tenant virtual networks. Accordingly, we presented a novel vswitch architecture which extends the benefits of multi-tenancy to the virtual switch, offering improved isolation and performance, at a modest additional resource cost. When used in the *shared* mode (only one extra core), with four vswitch compartments the forwarding throughput (in pps) is 1.5-2 times better than the Baseline. The tenant workloads (webserver and key-value stores) we evaluated also receive a 1.5-2 times performance (throughput and response time) improvement with *MTS*.

We believe that *MTS* is a pragmatic solution that can enhance the security and performance of virtual networking in the cloud. In particular, *MTS* introduces a way to schedule an entire core for tenant-specific network virtualization which has three benefits: (i) application and packet processing performance is improved; (ii) this could be integrated into pricing models to appropriately charge customers on-demand and generate revenue from virtual networking for example; (iii) virtual network and Host isolation is maintained even when the vswitch is compromised.

9 Acknowledgments

We thank our shepherd Leonid Ryzhyk and our anonymous reviewers for their valuable feedback and comments. We thank Ben Pfaff, Justin Pettit, Marcel Winandy, Hagen Woesner, Jean-Pierre Seifert and the Security in Telecommunications (SecT) team from TU Berlin for valuable discussions at various stages of this paper. The first author (K. T.) acknowledges the financial support by the Federal Ministry of Education and Research of Germany in the framework of the Software Campus 2.0 project nos. 01IS17052 and 01IS1705, and the "API Assistant" activity of EIT Digital. The third author (G. R.) is with the MTA-BME Information Systems Research Group.

References

[1] ALLCLAIR, T. Secure Container Isolation: Problem Statement & Solution Space. https://docs.google.com/document/d/1QQ5u1RBDLXWvC8K3pscTtTRThsOeBSts_imYEoRyw8A, 2018. Accessed: 05-01-2019.

[2] APACHE. ab - Apache HTTP server benchmarking tool. https://httpd.apache.org/docs/2.2/en/programs/ab.html. Accessed: 07-01-2019.

[3] ARNAUTOV, S., TRACH, B., GREGOR, F., KNAUTH, T., MARTIN, A., PRIEBE, C., LIND, J., MUTHUKUMARAN, D., O'KEEFFE, D., STILLWELL, M. L., GOLTZSCHE, D., EYERS, D., KAPITZA, R., PIETZUCH, P., AND FETZER, C. SCONE: Secure linux containers with intel SGX. In *Proc. Usenix Operating Systems Design Principles (OSDI)* (2016).

[4] AWS. Enhanced Networking on Linux. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html, 2018. Accessed: 24-01-2018.

[5] AZURE, M. Create a Linux virtual machine with Accelerated Networking. https://docs.microsoft.com/en-us/azure/virtual-network/create-vm-accelerated-networking-cli, 2018. Accessed: 24-01-2018.

[6] BATES, A., MOOD, B., PLETCHER, J., PRUSE, H., VALAFAR, M., AND BUTLER, K. On detecting co-resident cloud instances using network flow watermarking techniques. *Springer International Journal of Information Security* (2014).

[7] BEAUPRÉ, A. Updates in container isolation. https://lwn.net/Articles/754433, 2018. Accessed: 09-01-2019.

[8] BESS COMITTERS. BESS (Berkeley Extensible Software Switch). https://github.com/NetSys/bess, 2017. Accessed: 09-05-2017.

[9] BISHOP, M. A. *Introduction to computer security*, vol. 50. Addison-Wesley Boston, 2005.

[10] BOYD-WICKIZER, S., AND ZELDOVICH, N. Tolerating malicious device drivers in linux. In *Proc. Usenix Annual Technical Conference (ATC)* (2010).

[11] Broadcom Samples Thor, World's First 200G Ethernet Controller with 50G PAM-4 and PCIe 4.0. https://www.broadcom.com/company/news/product-releases/2367107. Accessed: 06-05-2019.

[12] COLP, P., NANAVATI, M., ZHU, J., AIELLO, W., COKER, G., DEEGAN, T., LOSCOCCO, P., AND WARFIELD, A. Breaking up is hard to do: Security and Functionality in a Commodity Hypervisor. In *Proc. ACM Symposium on Operating System Principles (SOSP)* (2011).

[13] COLUMBUS, L. Roundup Of Cloud Computing Forecasts And Market Estimates. https://www.forbes.com/sites/louiscolumbus/2018/09/23/roundup-of-cloud-computing-forecasts-and-market-estimates-2018/, 2017. Accessed: 09-01-2019.

[14] COSTIN, A. All your cluster-grids are belong to us: Monitoring the (in)security of infrastructure monitoring systems. In *Proc. IEEE Communications and Network Security (CNS)* (Sept 2015).

[15] CSIKOR, L., ROTHENBERG, C., PEZAROS, D. P., SCHMID, S., TOKA, L., AND RÉTVÁRI, G. Policy injection: A cloud dataplane dos attack. In *Proc. ACM SIGCOMM Posters and Demos* (2018).

[16] DALTON, M., SCHULTZ, D., ADRIAENS, J., AREFIN, A., GUPTA, A., FAHS, B., RUBINSTEIN, D., ZERMENO, E. C., RUBOW, E., DOCAUER, J. A., ALPERT, J., AI, J., OLSON, J., DECABOOTER, K., DE KRUIJF, M., HUA, N., LEWIS, N., KASINADHUNI, N., CREPALDI, R., KRISHNAN, S., VENKATA, S., RICHTER, Y., NAIK, U., AND VAHDAT, A. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *Proc. Usenix Networked Systems Design and Implementation (NSDI)* (2018).

[17] DPDK. Writing Efficient Code. https://doc.dpdk.org/guides/prog_guide/writing_efficient_code.html. Accessed: 06-01-2019.

[18] [PATCH] netdev-dpdk: Add new 'dpdkvhostuserclient' port type. https://mail.openvswitch.org/pipermail/ovs-dev/2016-August/321742.html. Accessed: 29-04-2019.

[19] EMMERICH, P., RAUMER, D., WOHLFART, F., AND CARLE, G. Performance characteristics of virtual switching. In *Proc. IEEE Conference on Cloud Networking* (2014).

[20] Endace DAG 10X4-P Datasheet. https://www.endace.com/dag-10x4-p-datasheet.pdf. Accessed: 07-01-2019.

[21] FIRESTONE, D. Vfp: A virtual switch platform for host sdn in the public cloud. In *Proc. Usenix Networked Systems Design and Implementation (NSDI)* (2017), pp. 315–328.

[22] FIRESTONE, D., PUTNAM, A., MUNDKUR, S., CHIOU, D., DABAGH, A., ANDREWARTHA, M., ANGEPAT, H., BHANU, V., CAULFIELD, A., CHUNG, E., CHANDRAPPA, H. K., CHATURMOHTA, S., HUMPHREY, M., LAVIER, J., LAM, N., LIU, F., OVTCHAROV, K., PADHYE, J., POPURI, G., RAINDEL, S., SAPRE, T., SHAW, M., SILVA, G., SIVAKUMAR, M., SRIVASTAVA, N., VERMA, A., ZUHAIR, Q., BANSAL, D., BURGER, D., VAID, K., MALTZ, D. A., AND GREENBERG, A. Azure accelerated networking: Smartnics in the public cloud. In *Proc. Usenix Networked Systems Design and Implementation (NSDI)* (2018).

[23] FRAZELLE, J. Hard multi-tenancy in kubernetes. https://blog.jessfraz.com/post/hard-multi-tenancy-in-kubernetes, 2018. Accessed: 09-01-2019.

[24] Google Compute Engine Pricing. https://cloud.google.com/compute/pricing#network, 2018. Accessed: 03-01-2019.

[25] Geneve: Generic Network Virtualization Encapsulation. https://tools.ietf.org/html/draft-ietf-nvo3-geneve-08. Accessed: 03-01-2019.

[26] The gVisor project. https://github.com/google/gvisor, 2018. Accessed: 09-01-2019.

[27] GOSPODAREK, A. The Rise of SmartNICs – offloading dataplane traffic to...software. https://youtu.be/AGSy51VlKaM, 2017. Open vSwitch Conference.

[28] HONDA, M., HUICI, F., LETTIERI, G., AND RIZZO, L. mswitch: a highly-scalable, modular software switch. In *Proc. ACM Symposium on SDN Research (SOSR)* (2015).

[29] HWANG, J., RAMAKRISHNAN, K., AND WOOD, T. Netvm: high performance and flexible networking using virtualization on commodity platforms. In *Proc. Usenix Networked Systems Design and Implementation (NSDI)* (2014).

[30] INTEL. Enabling NFV to deliver on its promise. https://www-ssl.intel.com/content/www/us/en/communications/nfv-packet-processing-brief.html, 2015.

[31] iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/. Accessed: 07-01-2019.

[32] JACOBSON, V. Congestion avoidance and control. In *ACM Computer Communication Review (CCR)* (1988).

[33] JAIN, R., AND PAUL, S. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communication Magazine 51*, 11 (2013).

[34] JIN, X., KELLER, E., AND REXFORD, J. Virtual switching without a hypervisor for a more secure cloud. In *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (HotICE)* (2012).

[35] JING, C. Zero-Copy Optimization for Alibaba Cloud Smart NIC Solution. http://www.alibabacloud.com/blog/zero-copy-optimization-for-alibaba-cloud-smart-nic-solution_593986, 2018. Accessed: 03-01-2019.

[36] The Kata Containers project. https://katacontainers.io, 2018. Accessed: 09-01-2019.

[37] KO, M., AND RECIO, R. Virtual ethernet bridging. http://www.ieee802.org/1/files/public/docs2009/new-hudson-vepa_seminar-20090514d.pdf. Accessed: 06-01-2019.

[38] KOPONEN, T., AMIDON, K., BALLAND, P., CASADO, M., CHANDA, A., FULTON, B., GANICHEV, I., GROSS, J., INGRAM, P., JACKSON, E., LAMBETH, A., LENGLET, R., LI, S.-H., PADMANABHAN, A., PETTIT, J., PFAFF, B., RAMANATHAN, R., SHENKER, S., SHIEH, A., STRIBLING, J., THAKKAR, P., WENDLANDT, D., YIP, A., AND ZHANG, R. Network virtualization in multi-tenant datacenters. In *Proc. Usenix Networked Systems Design and Implementation (NSDI)* (2014).

[39] KUTCH, P. PCI-SIG SR-IOV primer: An introduction to SR-IOV technology. *Intel application note* (2011), 321211–002.

[40] LÉVAI, T., PONGRÁCZ, G., MEGYESI, P., VÖRÖS, P., LAKI, S., NÉMETH, F., AND RÉTVÁRI, G. The Price for Programmability in the Software Data Plane: The Vendor Perspective. *IEEE J. Selected Areas in Communications* (2018).

[41] HowTo Launch VM over OVS-DPDK-17.11 Using Mellanox ConnectX-4 and ConnectX-5. https://community.mellanox.com/s/article/howto-launch-vm-over-ovs-dpdk-17-11-using-mellanox-connectx-4-and-connectx-5. Accessed: 09-01-2019.

[42] MELLANOX. Mellanox BlueField SmartNIC. https://bit.ly/2JaMitA, 2017. Accessed: 05-06-2018.

[43] Memcached. https://libmemcached.org/libMemcached.html. Accessed: 07-01-2019.

[44] MICROSOFT. Hyper-V Virtual Switch Overview. https://technet.microsoft.com/en-us/library/hh831823(v=ws.11).aspx, 2013. Accessed: 27-01-2017.

[45] MICROSOFT. SR-IOV VF Failover and Live Migration Support. https://docs.microsoft.com/en-us/windows-hardware/drivers/network/sr-iov-vf-failover-and-live-migration-support, 2017. Accessed: 03-01-2019.

[46] MOLNÁR, L., PONGRÁCZ, G., ENYEDI, G., KIS, Z. L., CSIKOR, L., JUHÁSZ, F., KŐRÖSI, A., AND RÉTVÁRI, G. Dataplane specialization for high-performance openflow software switching. In *Proc. ACM SIGCOMM* (2016).

[47] MULLER, A. OVS ARP Responder – Theory and Practice. https://assafmuller.com/2014/05/21/ovs-arp-responder-theory-and-practice/. Accessed: 06-01-2019.

[48] NEUGEBAUER, R., ANTICHI, G., ZAZO, J. F., AUDZEVICH, Y., LÓPEZ-BUEDO, S., AND MOORE, A. W. Understanding pcie performance for end host networking. In *Proc. ACM SIGCOMM* (2018).

[49] NEVES, M., LEVCHENKO, K., AND BARCELLOS, M. Sandboxing data plane programs for fun and profit. In *Proc. ACM SIGCOMM Posters and Demos* (2017).

[50] PALADI, N., AND GEHRMANN, C. Sdn access control for the masses. *Elsevier Computers & Security* (2019).

[51] PANDA, A., HAN, S., JANG, K., WALLS, M., RATNASAMY, S., AND SHENKER, S. Netbricks: Taking the v out of nfv. In *Proc. Usenix Operating Systems Design Principles (OSDI)* (2016).

[52] PANICKER, M. Enabling Hardware Offload of OVS Control & Data plane using LiquidIO. https://youtu.be/qjXBRCFhbqU, 2017. Open vSwitch Conference.

[53] PETTIT, J., PFAFF, B., STRINGER, J., TU, C.-C., BLANCO, B., AND TESSMER, A. Bringing platform harmony to vmware nsx. *ACM SIGOPS Operating System Review* (2018).

[54] PETTIT, J., PFAFF, B., WRIGHT, C., AND VENUGOPAL, M. OVN, Bringing Native Virtual Networking to OVS. https://networkheresy.com/2015/01/13/ovn-bringing-native-virtual-networking-to-ovs/, 2015. Accessed: 27-01-2017.

[55] PFAFF, B. Open vSwitch: Past, Present, and Future. http://openvswitch.org/slides/ppf.pdf, 2013. Accessed: 27-01-2017.

[56] PFAFF, B., PETTIT, J., KOPONEN, T., JACKSON, E., ZHOU, A., RAJAHALME, J., GROSS, J., WANG, A., STRINGER, J., SHELAR, P., AMIDON, K., AND CASADO, M. The design and implementation of Open vSwitch. In *Proc. Usenix Networked Systems Design and Implementation (NSDI)* (2015).

[57] RAM, K. K., COX, A. L., CHADHA, M., RIXNER, S., AND BARR, T. Hyper-switch: A scalable software virtual switching architecture. In *Proc. Usenix Annual Technical Conference (ATC)* (2013).

[58] RIZZO, L. Netmap: a novel framework for fast packet I/O. In *Proc. Usenix Annual Technical Conference (ATC)* (2012).

[59] RIZZO, L., AND LETTIERI, G. VALE, a switched ethernet for virtual machines. In *Proc. ACM CoNEXT* (2012).

[60] ROBIN G. Open vSwitch with DPDK Overview. https://software.intel.com/en-us/articles/open-vswitch-with-dpdk-overview, 2016. Accessed: 27-01-2017.

[61] SALTZER, J. H., AND SCHROEDER, M. D. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (1975), 1278–1308.

[62] SECURITYTWEEK. CSA's cloud adoption, practices and priorities survey report. http://www.securityweek.com/data-security-concerns-still-challenge, 2015. Accessed: 09-01-2019.

[63] SHAHBAZ, M., CHOI, S., PFAFF, B., KIM, C., FEAMSTER, N., MCKEOWN, N., AND REXFORD, J. Pisces: A programmable, protocol-independent software switch. In *Proc. ACM SIGCOMM* (2016).

[64] SMOLYAR, I., BEN-YEHUDA, M., AND TSAFRIR, D. Securing self-virtualizing ethernet devices. In *Proc. Usenix Security Symp.* (2015).

[65] STECKLINA, J. Shrinking the hypervisor one subsystem at a time: A userspace packet switch for virtual machines. In *Proc. ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE)* (2014).

[66] STONE, R. PCI SR-IOV on FreeBSD. `https://people.freebsd.org/~rstone/BSDCan_SRIOV.pdf`. Accessed: 06-01-2019.

[67] THE FAST DATA PROJECT. What is the Fast Data Project (FD.io)? `https://fd.io/about`, 2017. Accessed: 05-06-2018.

[68] THIMMARAJU, K., RÉTVÁRI, G., AND SCHMID, S. Virtual network isolation: Are we there yet? In *Proc. ACM Workshop on Security in Softwarized Networks: Prospects and Challenges* (2018).

[69] THIMMARAJU, K., SHASTRY, B., FIEBIG, T., HETZELT, F., SEIFERT, J.-P., FELDMANN, A., AND SCHMID, S. Taking control of sdn-based cloud systems via the data plane. In *Proc. ACM Symposium on SDN Research (SOSR)* (2018).

[70] TSENG, J., ET AL. Accelerating open vswitch with integrated gpu. In *Proc. ACM Workshop on Kernel-Bypass Networks* (2017).

[71] VANOVER, R. Virtual switching to become enhanced with Cisco and VMware announcement. `http://www.techrepublic.com/blog/data-center/virtual-switching-to-become-enhanced-with-cisco-and-vmware-announcement`, 2008. Accessed: 27-01-2017.

[72] VMWARE. VMware ESX 4.0 Update 1 Release Notes. `https://bit.ly/2sFTuTy`, 2009. Accessed: 05-06-2018.

[73] Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 network. `https://tools.ietf.org/html/rfc7348`. Accessed: 01-06-2016.

[74] ZHAO, Y. PCI: Linux kernel SR-IOV support. `https://lwn.net/Articles/319651/`, 2009. Accessed: 06-01-2019.

[75] ZHOU, Z., LI, Z., AND ZHANG, K. All your vms are disconnected: Attacking hardware virtualized network. In *Proc. ACM Conference on Data and Application Security and Privacy (CODASPY)* (2017).